Express Mail Label: EM049017355US
Date of Deposit: June 23, 1997

SYSTEM AND METHOD FOR MAXIMIZING USAGE OF COMPUTER RESOURCES IN SCHEDULING OF APPLICATION TASKS

Field of the Invention

This invention relates to scheduling of applications among processes in one or more associated computers; and, more particularly, to a scheduling system which implements a task schedule by setting operating system priorities for the processes working on queued activities to optimize usage of shared computer resources.

Background of the Invention

Scheduling of activities is needed when a computer is running multiple activities or applications. Assuming that each application or activity comprises more than one task, the tasks must be scheduled among available processes of the computer, often with the order of tasks being predetermined based upon the requirements of the application. For example, when doing a merge-sort operation, the tasks of sorting records into lists must be performed before the next task of merging the lists. The scheduling task becomes more challenging in a multi-processor

parallel computing environment, where multiple tasks may be run simultaneously by associated processes. For optimal usage of the available resources, processes should have waiting tasks queued for commencement as soon as previous tasks have been completed, with "wait states" being filled in with queued tasks.

In the past, load control has been used for multi-process scheduling. Under a load control scheduling scheme, only a subset of the total number of tasks are allowed to run at one time. If the processes for each of the subset of tasks all enter wait states (for example, pending the completion of a parallel-running task of the application by another process), the CPU will be unused throughout the duration of the wait states, even though there is more work queued. Scheduling of too few activities under the load control mechanism, therefore, frequently leads to underutilization of the CPU. On the other hand, if too many activities are allowed to run at once under a load control scheme, which is done under the assumption that all activities will not enter wait states at the same time, the ability to schedule among all of the tasks which are running is lost.

Another scheduling method which has been used in the prior art is priority-based scheduling for management of computer resources. Under a priority-based scheduling scheme, an operating system scheduler prioritizes the workload and schedules one task to be active at any given time. For example, on the AIX* (* Trademark of International Business Machines Corporation) operating system, if applications A, B and C are to be scheduled in alphabetical order,

and processes 1, 3 and 5 are working on A, 2 and 4 on B, and 6 on C, then processes 1, 3 and 5 have their operating system priorities set to 60, 2 and 4 to 61, and 6 to 62 (where lower process priority is more favorable). The prioritization scheme is effectively a resource utilization mechanism that does not perform scheduling of prioritized tasks among processes with the intent of running one or more applications as quickly and efficiently as possible.

When an activity to be scheduled does not parallelize into even amounts of work, neither load nor uniprocessor priority scheduling can maximize the application throughput. Database management systems, wherein the amount of work necessary for any task cannot be quantified in advance without detailed knowledge of the database and of the transactions to be performed thereon, defy scheduling by load or uniprocessor prioritization. Ideally, the scheduler must provide the ability to continue on to other activities related to the initial task when part of a parallel activity has been completed yet other related parts have not been completed.

What is desirable, therefore, is a dynamic priority scheduling mechanism for scheduling activities with multiple tasks among multiple processes, for minimizing unused CPU time.

It is therefore an objective of the present invention to implement scheduling at the task level.

It is additionally an objective of the present invention to provide multiple task scheduling which will minimize unused CPU time.

Still another objective of the invention is to provide coordination of activities among parallel computational resources to minimize unused CPU and optimize application run-time.

Summary of the Invention

These and other objectives are realized by the present invention wherein a task schedule is enforced among multiple processes by setting process priorities based upon which tasks are running on which processes and based upon the task schedule. The task scheduling may be provided by a local or global scheduler which uses application information to prioritize tasks. The task schedule, or priority list, is provided at Local Activity Schedulers, which schedule the activities for their local execution elements/nodes. Execution of activities locally ~~are~~ is performed by any number of processes that reside in each execution element. These processes are assigned operating system priorities by the respective Local Activity Scheduler based on their assigned activities for execution and the task schedule.

Brief Description of the Drawings

The invention will now be described in detail with specific reference to the attached drawings wherein:

Figure 1 provides a schematic illustration of a parallel processing system utilizing one embodiment of the present invention.

Figure 2 illustrates the Activity Priority list maintained in accordance with the present invention.

Figure 3 illustrates the Local Activity Scheduler list maintained by the Global Activity Scheduler of the present invention.

Figure 4 illustrates the Activity-Process Correspondence Table maintained in accordance with the present invention.

Figure 5 provides a flow chart representative of the operations of the Global Activity Scheduler of the present invention.

Figure 6 provides a flow chart representative of the operations of the Local Activity Schedulers of the present invention.

Figure 7 provides a Gantt Chart of prior art scheduling of 4-way parallel activities.

Figure 8 provides a Gantt Chart of scheduling of 4-way parallel activities in accordance with the present invention.

Description of the Preferred Embodiment

One embodiment of the inventive multiple activity scheduling system for a parallel processing environment is illustrated in

Figure 1. As shown therein, a Global Activity Scheduler 10, utilizing information r ceived from the applications as provided via the Application Coordinator 11, provides a prioritized schedule of tasks or activities along communication links to nodes 16-19. Each node is provided with a Local Activity Scheduler 12-15 which schedules each of the its associated processes, 102-109. The schedule information may be regularly updated based upon incoming activities to be scheduled and based upon information provided by continual monitoring of the resources at the nodes. As illustrated, communications between the entities are bi-directional, with the Local Activity Schedulers continuously reporting process information to the Global Activity Scheduler directly or via updates through the Application Coordinator, as will be further detailed below. While the system has been illustrated to include four nodes, each having two processes, it will be apparent that the present invention can be applied to a system having any number of nodes in communication with a Global Activity Scheduler, wherein each node may have any number of associated processes. Each node in the system is necessarily provided with a dedicated Local Activity Scheduler, which may or may not be physically located at the node. If the Local Activity Scheduler which is associated with a particular node is not physically located at that node, it is understood that the Local Activity Scheduler would be in constant communication with the operating system at the node. The Local Activity Scheduler is responsible for establishing the operating system priorities for

implementing the task schedule at the node. In an alternative embodiment, the Local Activity Scheduler may itself stablish the task schedule, if no global entity is available or required, as further detailed below.

In the illustrated embodiment, for each activity to be scheduled, an activity ID is created. The activity ID can be the application command string, the user ID running the command, or an ID created by the application or by some other process at the Application Coordinator or Global Activity Scheduler. In addition, each process in a node has a process ID. When a process begins or ends its work on a task, it reports its activity ID and process ID to the Local Activity Scheduler process, which in turn reports the activity ID directly to the Global Activity Scheduler in a Begin-End Task message or indirectly via future updates from the Application Coordinator. The Global Activity Scheduler is provided with knowledge of which applications are in the system. Then the Global Activity Scheduler creates a schedule based on a scheduling algorithm (or uses a pre-determined schedule) for the application tasks and forwards this schedule to each Local Activity Scheduler associated with each node of the parallel computer.

The Local Activity Schedulers are each responsible for tracking which processes are working on which applications at their node. Using this knowledge and the task schedule (hereinafter referred to as the Activity Priority list), each Local Activity Scheduler determines the priorities for each of the processes on its node, and directs the local operating system to set the process

priorities for optimal execution of the prioritized activities.

All of the processes working on tasks for the same activity will get the same priority, with each activity being assigned a unique priority. When no more unique priorities exist, the activities scheduled at the end can all be given the least favored priority. When all the processes in the highest prioritized activities are not using CPU, the operating system will have the activity with the second highest priority use the CPU, and so on, to thereby limit the amount of unused CPU time.

For the Figure 1 embodiment, Figures 2 and 3 provide representative examples of two internal structures which would be dynamically maintained at the Global Activity Scheduler, specifically the Activity Priority list and the Local Activity Scheduler list. The Local Activity Scheduler list of Figure 3 comprises a list of all active Local Activity Schedulers in the system which are under the control of the Global Activity Scheduler (for example, at all nodes of a partition). As shown in Figure 3, the list includes the Local Scheduler ID along with its location address. The Local Activity Scheduler list is maintained for utilization when broadcasting priority information and is continually updated by communications received from the Local Activity Schedulers.

The Activity Priority list of Figure 2 is the task schedule which is derived from communication with the applications (from the Application Coordinator in the Figure 1 implementation), as to which activities are active in the system. The Activity Priority

list provides the activity IDs in priority order. Reception of Begin and End application messages from the Application Coordinator allows the scheduling program at the Global Activity Scheduler, or at the Local Activity Scheduler in the alternative embodiment, to maintain the Activity Priority list. The priorities of activities on the list can be determined by utilizing any number of parameters related to the activities to be scheduled. Examples of relevant parameters include the relative importance of the user activities (as provided by the user or a programmer), time deadlines, user expense guidelines, resource requirements, etc. Any message reception from the Application Coordinator triggers the relevant scheduling program out of its wait state to generate an updated Activity Priority list. In addition, completion of an activity, as communicated from the processes at a node, will cause removal of the activity from the Activity Priority list, and updating of the schedule. In the Figure 1 embodiment, the Global Activity Scheduler broadcasts the Activity Priority list to all Local Activity Schedulers either immediately upon generation of a new list or at periodic intervals.

At the Local Activity Scheduler, the latest version of the Activity Priority list is maintained, as communicated from the Global Activity Scheduler in the Figure 1 embodiment, or as derived locally in the alternative embodiment. In addition, the Local Activity Scheduler maintains an Activity-Process Correspondence table, as shown in Figure 4. The Activity-Process Correspondence table reflects the assignment of activities at the node to the

node's processes, along with the respective priorities of the activities. This information may be obtained directly from the processes themselves, under a task registration protocol, or indirectly, for example from the database monitor of a database on which processes are performing tasks. In the Figure 4 illustration, assuming a node having five available processes, the Activity IDs are paired with process IDs, with the respective activity priority assignments listed with the activity-process pairings. The priority assignments found on the Activity-Process Correspondence table are assigned by the Local Activity Scheduler based upon the Activity Priority list.

Figure 5 provides a representative process flow of the operations performed by the Global Activity Scheduler of the Figure 1 embodiment. At box 50, a communication packet is received at the Global Activity Scheduler, and, in step 51, the packet is analyzed to determine if the message is from the Local Activity Scheduler or from the Application Coordinator. If the message comprises a communication regarding processes from the Local Activity Scheduler, the Activity Scheduler list is updated to reflect the available processes, at step 52. If the message is from the Application Coordinator regarding completed tasks or tasks to be commenced, the scheduling program in invoked and the Activity Priority list updated at step 53. Depending upon the preferred programming order of operations, the Global Activity Scheduler may automatically send the updated Activity Priority list to all Local Activity Schedulers, as shown at step 54, or may wait until a pre-

set time interval has elapsed, as indicated by decision box 55, and then send the list. The Global Activity Scheduler then waits for the next communication packet, as shown at step 56.

Figure 6 illustrates the representative process flow of operations conducted by the Local Activity Scheduler of the Figure 1 embodiment. Upon receipt of a communication packet at 60, the Local Activity Scheduler determines, as indicated by decision box 61, whether the communication is from the Global Activity Scheduler or from one of its activity processes. If the message is from the Global Activity Scheduler, the message will contain an Activity Priority list to replace the previously-communicated list, as reflected at step 62. As noted above, the Local Activity Scheduler could, in an alternative embodiment, be the entity that establishes the task schedule, and would therefore automatically update its Activity Priority list. If the message is from an activity process, presumably either a Begin or End task message, then the Local Activity Scheduler updates the Activity-Process Correspondence table at step 63. In addition, at this juncture, the Local Activity Scheduler may communicate task-related messages (not shown) to the Global Activity Scheduler. Since communications between nodes and the applications/Application Coordinator regarding task commencement and completion are well known and need not be altered to implement the present invention, and since the Application Coordinator necessarily relays such information to the Global Activity Scheduler, it is not strictly necessary to incorporate the redundant step of the Local Activity Scheduler

communicating task commencement and completion messages to the Global Activity Scheduler.

Simultaneously with, or subsequent to, the appropriate updating of the Activity-Process Correspondence table, the Local Activity Scheduler assigns priorities, at step 64, and directs the local operating system to set the process priorities for execution of tasks at the node, at step 65. Finally, the Local Activity Scheduler enters a waiting state at step 66, awaiting receipt of the next communication packet.

By dynamically assigning priorities and allowing the local processes to move from one task to the next highest priority task without waiting, the system maximizes utilization of the CPU resources at each node. Figure 7 shows a Gantt Chart of Random scheduling of three 4-way parallel activities. In the prior art example, three parallel activities (A, B and C) are distributed across four processes or machines and are active at time zero. When each of the respective parallel parts are complete, the activity associated with the task is complete. The Random chart shows that each of the tasks is complete at time 30, and that the average time for completion is 30. This is because at least on parallel part of each activity was not able to begin until time 20. On the other hand, when using the inventive system, as illustrated on the Gantt chart of Figure 8, the average time to complete an activity would be 20, resulting in a speed-up of 33.3%, due to the parallel scheduling of activities among the processes.

It is to be noted that in a parallel environment wherein the

queued activities to be scheduled are similar in size and arrival time and are performing similar tasks, in terms of resource requirements, such as periodic display of fixed amounts of data retrieved from an outside source or database operations wherein the database is divided equally among the available processes, the activities of the Global Activity Scheduler could be performed by the Local Activity Schedulers, which would generate virtually identical schedules, thereby obviating the need for coordination of those schedules at the "global" level.

The invention has been described with reference to several specific embodiments. One having skill in the relevant art will recognize that modifications may be made without departing from the spirit and scope of the invention as set forth in the appended claims.